Конспект лекции
Фреймворки логгирования: логгирование в Java

Цель и задачи лекции

Цель – научиться сопровождать программный код выводом сообщений в лог.

Задачи:

1. Определить, что такое лог
2. Ознакомиться с основными фреймворками логгирования
3. Узнать, как использовать логгирование в программном коде

План занятия

1. Логгирование и требования к нему
2. Обзор основных фреймворков логгирования
3. Использование Log4j и SLF4j

Определение

Логи (лог-файлы) — это файлы, содержащие системную информацию работы сервера или компьютера, в которые заносятся определенные действия пользователя или программы.

Их предназначение — протоколирование операций, выполняемых на машине, для дальнейшего анализа администратором. Регулярный просмотр журналов позволит определить ошибки в работе системы в целом, конкретного сервиса или сайта (особенно скрытые ошибки, которые не выводятся при просмотре в браузере), диагностировать злонамеренную активность, собрать статистику посещений сайта.

Типы логов и требования к ним

Принципиально логи нужны не только для отслеживания программных ошибок – точно так же можно отслеживать и ошибки (а то и целенаправленные действия) пользователей. Формально протокол действий приложения можно разделить на несколько частей. Если говорить об очень общем делении, то таких частей будет две – что происходит в приложения с точки зрения бизнес-пользователя и что происходит в приложение с точки зрения разработчика (программиста). Если смотреть в несколько другой плоскости – можно поделить все действия на системные (SYSTEM), безопасности (SECURITY) и приложения (APPLICATION, BUSINESS).

Из всего перечисленного нас – разработчиков – интересует прежде всего то, что происходит в системе с точки зрения именно разработчика. Что же касается классификации происходящих событий – SYSTEM/SECURITY/APPLICATION – разработчикам приходится иметь дело со всеми ними.

Дальше. Простой вопрос – а сколько информации нам нужно? Такой же простой ответ – чем больше, тем лучше! – не годится. Ибо чем больше мы выводим информации о происходящем в системе – тем больше процессорного времени тратится на эти бесполезные с точки зрения конечного пользователя действия, и тем меньше остается собственно на работу. Как пример, интенсивный вывод в лог(ошиблись в конфигурации) может замедлить работу системы более чем в 50 раз. Вот тут на самом деле есть противоречие – с точки зрения пользователя логов должно быть в минимальном объеме, а с точки зрения разработчика – в максимальном.

Следующее очевидное требование – вывод только той информации, которая нам нужна, и в том виде, в котором она нам нужна. Т.е. необходима возможность управлять форматом вывода – что именно выводить, мы и так контролируем. Причем крайне желательно, чтобы такая настройка была отделена от приложения – чтобы ее можно было менять, не трогая программного кода.

Унифицированные системы логирования

Рассмотрим три основных фреймворка для ведения лога – библиотека Log4J, пакет java.util.logging в JavaSE и библиотека Logback. Также рассмотрим Apache Commons Logging и Simple Logging Facade for Java. Эти фреймворки являются прослойками между системой логирования и самим приложением.

Log4J

Фреймворк Apache Log4J появился первым из всех рассматриваемых. Изначально этот фреймворк был хорошо архитектурно проработан, потому он быстро завоевал популярность. Возможно, определенную роль сыграл также факт, что других просто не было.

java.util.logging

Пакет java.util.logging появился в JavaSE в версии 1.4, в 2001 году. К этому моменту уже существовал Log4J, судя по всему, от него и отталкивались.

Соответственно, в какой-то момент сложилась не очень приятная ситуация. В части приложений используется Log4J, в части – пакет java.util.logging. И все бы ничего, но возникает вопрос: а что делать разработчикам библиотек? Им тоже нужно использовать логирование. На какую из систем ориентироваться? API этих фреймворков различается, связывание происходит на этапе компиляции, просто так фреймворк логирования не заменить.

Выход был найден достаточно простой. Раз у нас используются два разных фреймворка для одной цели – их надо унифицировать. Т.е. написать прослойку (фактически, адаптер), которая будет скрывать от разработчика, какой реально фреймворк он использует. Он будет звать прослойку, а она – конкретный фреймворк. Так появился Apache Commons Logging.

Apache Commons Logging

Фреймворк Apache Commons Logging предназначен для абстрагирования разработчика от конкретного фреймворка логирования. Он предоставляет некоторый унифицированный интерфейс, транслируя его вызовы в использование конкретных возможностей фреймворков. Ключевым тут является слово "унифицированный". Оно означает минимальное доступное покрытие, т.е. отсутствие, например, специфических возможностей Log4J.

Commons Logging абстрагирует следующие фреймворки:Log4J, java.util.logging, Avalon LogKit, Lumberjack.

Появившись, Commons Logging стал спасением для разработчиков библиотек общего назначения. В результате этого он используется необычайно широко. Однако у него есть ряд существенных недостатков, с которыми приходилось мириться.

Фреймворк является минимальным – не поддерживает очень полезных специфических возможностей абстрагируемых фреймворков.

Фреймворк никак не занимается инициализацией и конфигурированием конкретных фреймворков логирования – это остается на самом разработчике. Но это все равно лучше, чем было до него – при смене фреймворка логирования нужно изменить только код инициализации.

Самое неприятное – при некоторых условиях у этого фреймворка есть проблемы с загрузчиком классов.

Если с первыми двумя неудобствами еще можно было как-то жить, то третье осложняло жизнь очень серьезно. В немалой степени из-за этого и появился следующий фреймворк – SLF4J.

SLF4J (Simple Logging Facade for Java)

По своей сути SLF4J является тем же, чем и Commons Logging – абстрагирующим фреймворком логирования. Однако у него есть ряд важных отличий:

- SLF4J является более продвинутым, нежели Commons Logging – набор поддерживаемых им возможностей конечных фреймворков шире. Если где-то конкретный конечный фреймворк не поддерживает каких-либо возможностей – делается их имитация.
- SLF4J поддерживает (абстрагирует) большее количество фреймворков логирования, чем Commons Logging – это java.util.logging, Log4J, Commons Logging, Logback (он вообще является реализацией интерфейсов самого SLF4J).
- SLF4J является приемником для java.util.logging, Log4J, Commons Logging – т.е. его можно подключить "под" эти фреймворки, так, что вывод через них будет перенаправляться в SLF4J. Единственное ограничение – невозможность работы в схеме Log4J >> SLF4J >> Log4J.
- Проблем с загрузчиком классов у него нет

Устроен SLF4J просто, в некоторой степени даже элегантно. Есть общая часть библиотеки, API. И есть несколько библиотек, каждая из которых реализует свою схему – SLF4J >> Log4J, SLF4J >> Commons Logging, SLF4J >> java.util.logging, SLF4J >> NOP, SLF4J >> Simple. Эти библиотеки подключаются простым помещением их в classpath. Сделано это следующим образом – каждая из дополнительных библиотек содержит класс с определенным именем (на самом деле классов несколько, но это неважно). Этот класс является, фактически, фабрикой для создания всех сущностей SLF4J на основе конкретного фреймворка.

В целом SLF4J является более удачным, нежели Commons Logging, потому в последнее время наблюдается тенденция перехода на него. Кстати, у этого фреймворка тот же автор, что и у Log4J.

Logback

Logback концептуально является наследником Log4J. Что неудивительно, ибо автор у них один – вместе с SLF4J. И опять-таки неудивительно, что Logback при этом является реализацией интерфейсов SLF4J, т.е. максимально к нему приближен. Что немаловажно с точки зрения прежде всего производительности.

От Log4J Logback взял все хорошее, что там было. То есть – практически всё. По используемым понятиям они похожи до степени смешения.

Log4j

Добавление зависимости

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.6.1</version>
</dependency>
```

Конфигурация

Конфигурирование log4j основано на файле log4j.xml.

Вывода лога в консоль

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="debug" name="baeldung" packages="">
    <Appenders>
        <Console name="stdout" target="SYSTEM_OUT">
            <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss} %p %m%n"/>
        </Console>
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="STDOUT"/>
        </Root>
    </Loggers>
</Configuration>
```

Вывод лога в файл

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration status="debug" name="baeldung" packages="">
    <Appenders>
        <File name="fout" fileName="baeldung.log" append="true">
            <PatternLayout>
                <Pattern>%d{yyyy-MM-dd HH:mm:ss} %-5p %m%nw</Pattern>
            </PatternLayout>
        </File>
```

```
    </Appenders>
    <Loggers>
        <Root level="error">
            <AppenderRef ref="stdout"/>
            <AppenderRef ref=fout""/>
        </Root>
    </Loggers>
</Configuration>
```

Асинхронное логгирование

```
<AsyncRoot level="DEBUG">
    <AppenderRef ref="stdout" />
    <AppenderRef ref="fout"/>
</AsyncRoot>
```

Программный код

```
import org.apache.logging.log4j.Logger;
import org.apache.logging.log4j.LogManager;

public class Log4jExample {

    private static Logger logger = LogManager.getLogger(Log4jExample.class);

    public static void main(String[] args) {
        logger.debug("Debug log message");
        logger.info("Info log message");
        logger.error("Error log message");
    }
}
```

Рекомендуется использовать логгирование для обработки исключений

```
try {
    // Here some exception can be thrown
} catch (Exception e) {
    logger.error("Error log message", throwable);
}
```

SLF4J

Добавление зависимости

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.7</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.7</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
```

```
    <artifactId>log4j-slf4j-impl</artifactId>
    <version>2.7</version>
</dependency>
```

Программный код

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class Log4jExample {

    private static Logger logger = LoggerFactory.getLogger(Log4jExample.class);

    public static void main(String[] args) {
        logger.debug("Debug log message");
        logger.info("Info log message");
        logger.error("Error log message");
    }
}
```

Литература и ссылки

1. http://www.skipy.ru/useful/logging.html
2. https://www.baeldung.com/java-logging-intro
3. https://www.baeldung.com/slf4j-with-log4j2-logback
4. https://hostiq.ua/wiki/log/

Вопросы для самоконтроля

1. Для чего нужен лог файл?
2. Какие основные библиотеки логгирования вы знаете?
3. Зачем нужно логгирование программы?
4. Нужно ли логгировать исключения?